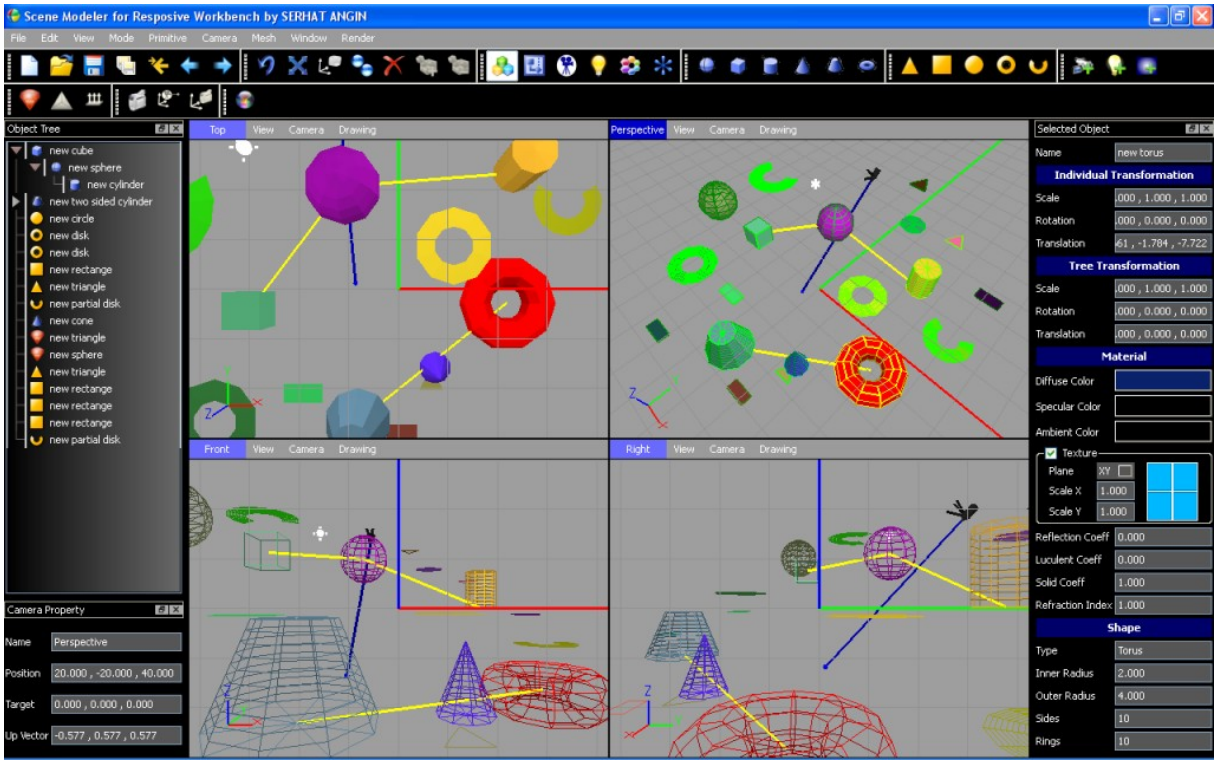
SCENE MODELER FOR RESPONSIVE WORKBENCH

by

Serhat ANGIN



Submitted to the Department of Computer Engineering
in partial fulfillment of the requirements
for the degree of
Bachelor of Science
in
Computer Engineering

Boğaziçi University
June 2009

Table of Contents

Abstract ……..... ……...4
1. INTRODUCTION……..... ……...4
2. DESIGN……..... ……...4
2.1 STORAGE CLASSES……..... ……....5
2.1.1 Point3 Class……..... ……...5
2.1.2 Vector3 Class……..... ……...5
2.1.3 Vertex Class……..... ……...5
2.1.4 Triangle Class……..... ……...5
2.1.5 Shape Class……..... ……...6
2.1.6 TriangleMesh Class……..... ……...6
2.1.8 Cube Class……..... ……...6
2.1.9 Sphere Class……..... ……...6
2.1.10 Cylinder Class……..... ……...7
2.1.11 Cone Class……..... ……...7
2.1.12 TwoFacetCylinder Class……..... ……...7
2.1.13 Torus Class……..... ……...7
2.1.14 TriangleShape Class……..... ……...7
2.1.15 Rectangle Class……..... ……...7
2.1.16 Circle Class……..... ……...7
2.1.17 Disk Class……..... ……...7
2.1.18 PartialDisk Class……..... ……...8
2.1.19 Rotation4 Class……..... ……...8
2.1.20 Transform Class……..... ……...8
2.1.21 Camera Class……..... ……...8
2.1.22 TRadiance Class……..... ……...8
2.1.23 Material Class……..... ……...8
2.1.24 Light Class……..... ……...8
2.1.23 Primitve Class……..... ……...8
2.1.24 GfxObject Class……..... ……...8
2.1.25 Scene Class……..... ……...9
2.2 FUNCTIONAL CLASSES……..... ……...9
2.2.1 EditingCamera Class……..... ……...9
2.2.2 SMMenuHandler Class……..... ……...9
2.2.3 ExtrudeDialog Class……..... ……...9
2.2.4 CanvasGrid Class……..... ……10
2.2.5 SMCanvasWindow Class……..... ……10
2.2.6 SMCanvasWidget Class……..... ……10
2.2.7 SMIODevice Class……..... ……10
2.2.8 RawDevice Class……..... ……10
2.2.9 StlDevice Class……..... ……11
2.2.10 VrmlDevice Class……..... ……11
2.2.11 CameraTool Class……..... ……11
2.2.12 TranslationTool Class……..... ……11
2.2.13 RotationTool Class……..... ……11
2.2.14 ScaleTool Class……..... ……11
2.2.15 CameraPropertiesController Class……..... ……11

2.2.16 ObjectTreeController Class……...……11
2.2.17 ObjectPropertyController Class……. ...……11
3. USER INTERFACE…….. ...……12
3.1 USER INTERFACE DESIGN……...……12
3.1.1 Menus……..... ...……13
3.1.1.1 File Menu……..... ...……13
3.1.1.1.1 New Action……..... ...……13
3.1.1.1.2 Open Action……... ...……13
3.1.1.1.3 Save Action……... ...……13
3.1.1.1.4 Save As Action…….. ...……13
3.1.1.1.5 Merge Action……. ...……13
3.1.1.1.6 Import Action……...……13
3.1.1.1.7 Export Action……...……13
3.1.1.1.8 Exit Action……..... ...……13
3.1.1.2 Edit Menu……..... ...……13
3.1.1.2.1 Rotate Action……. ...……14
3.1.1.2.2 Scale Action……... ...……14
3.1.1.2.3 Translate Action……...……14
3.1.1.2.4 Duplicate Action……..... ...……14
3.1.1.2.5 Delete Action……. ...……14
3.1.1.2.6 Add Child Action……... ...……14
3.1.1.2.7 Remove Child Action……..... ...……14
3.1.1.3 View Menu……...……14
3.1.1.5 Mode Menu……..... ...……15
3.1.1.5.1 Single Object Action…….. ...……15
3.1.1.5.2 Object Tree Action……..... ...……15
3.1.1.5.3 Camera Action……... ...……15
3.1.1.5.4 Light Action……... ...……16
3.1.1.5.5 Face Action……..... ...……16
3.1.1.2.6 Vertex Action……...……16
3.1.1.6 Primitive Menu……. ...……16
3.1.1.6.1 New Camera Action…….. ...……16
3.1.1.6.2 Point Light Action……. ...……16
3.1.1.6.3 Area Light Action…….. ...……16
3.1.1.6.4 3D objects submenu……... ...……16
3.1.1.6.5 2D objects submenu……... ...……16
3.1.1.7 Camera Menu……..... ...……16
3.1.1.7.1 Edit Camera Action……...……17
3.1.1.7.2 Move Target Action……... ...……17
3.1.1.7.3 Move Camera Action……...……17
3.1.1.8 Mesh Menu……..... ...……17
3.1.1.8.1 Convert To Mesh Action……...……17
3.1.1.8.2 Split Face Action……...……17
3.1.1.8.3 Extrude Face Action…….. ...……17
3.1.1.9 Window Menu…….. ...……17
3.1.1.10 Render Menu……...……18
3.1.2 Tool Bars……...……18

3.1.2.1 File Toolbar…….....……18
3.1.2.2 Edit Toolbar……..... ...……18
3.1.2.3 Mode Toolbar……...……19
3.1.2.4 3D objects Toolbar……...……19
3.1.2.5 2D objects Toolbar……...……19
3.1.2.6 Primitive Toolbar…….. ...……19
3.1.2.7 Mesh Toolbar……..... ...……19
3.1.2.8 Camera Toolbar……..... ...……19
3.1.2.9 Render Toolbar……. ...……20
3.1.3 Canvas Views……..... ...……20
3.1.3.1 Layout of a canvas view……...……21
3.1.3.1.1 View Menu Of Canvas View……. ...……21
3.1.3.1.2 Camera Menu Of Canvas View……. ...……21
3.1.3.1.3 Drawing Menu Of Canvas View……...……22
3.1.3.1.4 Drawing Region in Canvas View….. ...……22
3.1.4 Dockable Windows…….. ...……22
3.1.4.1 Object Tree Window……...……23
3.1.4.2 Camera Property Window……..... ...……23
3.1.4.3 Object Property Window…….. ...……24
3.1.5 Dialogs……. ...……25
3.1.5.1 Edit Camera Dialog……...……25
3.1.5.2 Extrude Face Dialog……...……26
3.1.5.3 Render Options Dialog……...……26
4. FORMATS…….....……26
4.1 VRML 2.0 FORMAT……. ...……26
4.1.1 Material Representation in VRML…….. ...……27
4.1.2 Shape Representation in VRML…….. ...……27
4.1.2.1 Sphere Representation…….. ...……27
4.1.2.2 Cylinder Representation……...……28
4.1.2.3 Cone Representation……. ...……28
4.1.2.3 Cube Representation……. ...……28
4.1.2.3 Other Shapes Representation……..... ...……28
4.1.3 GfxObject Representation in VRML….. ...……29
4.1.4 Camera Representation in VRML……...……29
4.1.5 Light Representation in VRML…….... ...……30
4.1.6 Parent-Child Hierarch in VRML……...……30
4.2 RAW FORMAT……...……31
4.3 STL FORMAT……..... ...……31
5 CONCLUSION…….....……32
6 DEVELOPMENT ENVIRONMENT……...……32
7 GLOSSARY…….....……32
8 REFERENCES…….....……33

Abstract: **3D scene creation and manipulation tools are widely used in many areas. In some areas, 3D modeling tools used to design new products and some other areas these tools are used to create virtual scene to virtual reality applications. For example, in the automotive sector designers uses 3D tools to design new car and in the game sector, modeler use these tools to create virtual environments to their games. 3D modeling tools used to design new products are scale sensitive this means the dimensions used in modeling steps are realistic. While creating virtual scenes for virtual reality applications, dimensions can be ignored so tools used for this aim do not guarantee the dimensions. At result, all 3D tools are designer for different purposes. In this project we designed and implement our 3D scene creation and manipulation tool for Responsive Workbench project. Mainly using our Scene Modeler, we can create and manipulate scenes for Responsive Workbench.**

1. INTRODUCTION

This document is prepared to explain design and development steps of scene modeler. In the first part, design, data structures and class information used to implement scene modeler explained. In the data structures part, an explanation for data types and structures to keep scene components included. Storing objects, a camera or a light briefly explained in this part. User interface design and program functionalities mentioned under the title user interface. The main window layout of program, scene views, windows, menu bar and tool bars are presented with helpful screenshots under user interface layout title. All functionalities are explained with descriptions are also included in user interface part. Fallowing part after user interface title is algorithms part. In this part some specific algorithms are described. After this part, Formats title is introduced. Some 3D file formats supported by scene modeler for responsive workbench are described under formats title. After all we summarize our work in the conclusion part.

2. DESIGN

In this section, we describe data structures and classes of scene modeler. We can categorize classes as storage and functional classes. In storage classes, we store objects, points, faces, cameras and some other components of scene. Data structures are represented into storage classes. Storage classes represent the core structure of our program. There exists a geometric hierarchy between storage classes. Basically it starts point, vector and vertex definitions and creates more complex structures including basic ones. One or more basic items build up complex structures. For example, three vertexes create a triangle and triangle creates a triangle-meshed shape. Some parametric geometric 3D or 2D shapes are also represented as data structures in scene modeler. In additional to geometric hierarch between storage classes, some other properties of geometric objects are also stored in core classes. Material and transform properties of geometric objects are stored in the material and transform classes. Beside storage classes about geometric objects, there exist some other classes represents other components of scene description like light and camera classes.

Functional classes are responsible to manipulate and create storage classes. Some functional classes have user interface items like form, window to interact with user. These classes are responsible for handling user inputs and changing user interface states according to program state. For example, these classes enable and disable actions in terms of the program mode. Some other functional classes implement some geometric 3D operations and transformations. One example for this part is rotation class that rotates geometric objects in terms of mouse events.

We can divide this part into two subparts as information about storage classes also called data structures and information about functional classes.

2.1 STORAGE CLASSES

We can categorize storage classes as classes of representation of geometric terms, classes of properties of geometric representations, and some other environment components for scene description. In the first category, classes that have geometrical meaning are included. Basic classes for this category are vector3, point3 and vertex3 classes. Vector3 type represents direction with x, y and z components. Point3 class represent a single point in the three dimensional space with x, y, and z coordinates. Vertex3 classes means with same geometric representation with point3, but it is special for that it can be a part of a triangle. In the second level after unit geometric parameters, we introduce the concept of face. In our work, we use face as triangles since it is easy usable and not complex structure. If scene modeler encourages a face that is not triangle, it triangulates this face with a specific algorithm. One after level is shape concept. Shape is an abstract class and we derive special geometric shapes from this abstract class. First commonly used shape is triangle mesh. It contains an array of vertex and triangles. Other shapes are representation of some parametric geometric objects like sphere. In the second category of classes, some classes are representation of a property of 3D geometric objects. Material class is first example for this category. Material class keeps some properties of material that geometric objects can have like diffuse color. Third category of classes includes light and camera classes that are other component of scene.

2.1.1 Point3 Class:

As we mention previously that Point3 class is representation of a point in 3D space with x, y and z coordinates. It has some useful operators like “*”, “-”, “*=”, “= =”. These operators used for some point3 manipulations.

2.1.2 Vector3 Class:

Vector3 is a representation of a unit vector in geometric terms. Unit vector means a direction vector whose length is equal to 0. It has x, y, z components. In addition to Point3 class’s operators, it has vector dot product and vector cross product operators. These operators are commonly used in the geometric calculations. It has also “normalize” function it divides x, y and z components with its length.

2.1.3 Vertex Class:

This class derives from Point3 class. Vertex3 is specific for points that are belonged by a triangle. In addition to Point3 class functionalities it has “selected” flag. This flag is true when a vertex is selected. In the vertex-editing mode, user can select and edit vertex with using this flag.

2.1.4 Triangle Class:

This class is introduction to face concept in Scene Modeler. It contains three vertex called a, b, and c, and a normal vector3 representing its normal and a point3 representing its midpoint. Vertexes of a triangle are references this means more than one face can reference a single copy of vertex, than if a value of vertex is changed all triangles referencing this vertex are automatically updated. It has also selection flag, in the face-editing mode an individual face is selectable and editable with some mouse interactions. In the constructer, vertex reference parameters taken and midpoint and normal are calculated. If a vertex of triangle are edited in some way, user can call “ restoreNormalAndMidPoint() ” function that recalculates midpoint and normal.

2.1.5 Shape Class:

Shape class is an abstract class for geometric shapes like triangle mesh, sphere and some other parametric geometric shapes. All shapes derived from can copy itself into a triangle mesh object meaning that they can triangulate themselves. ***Figure 1*** show classes derived from this abstract class.

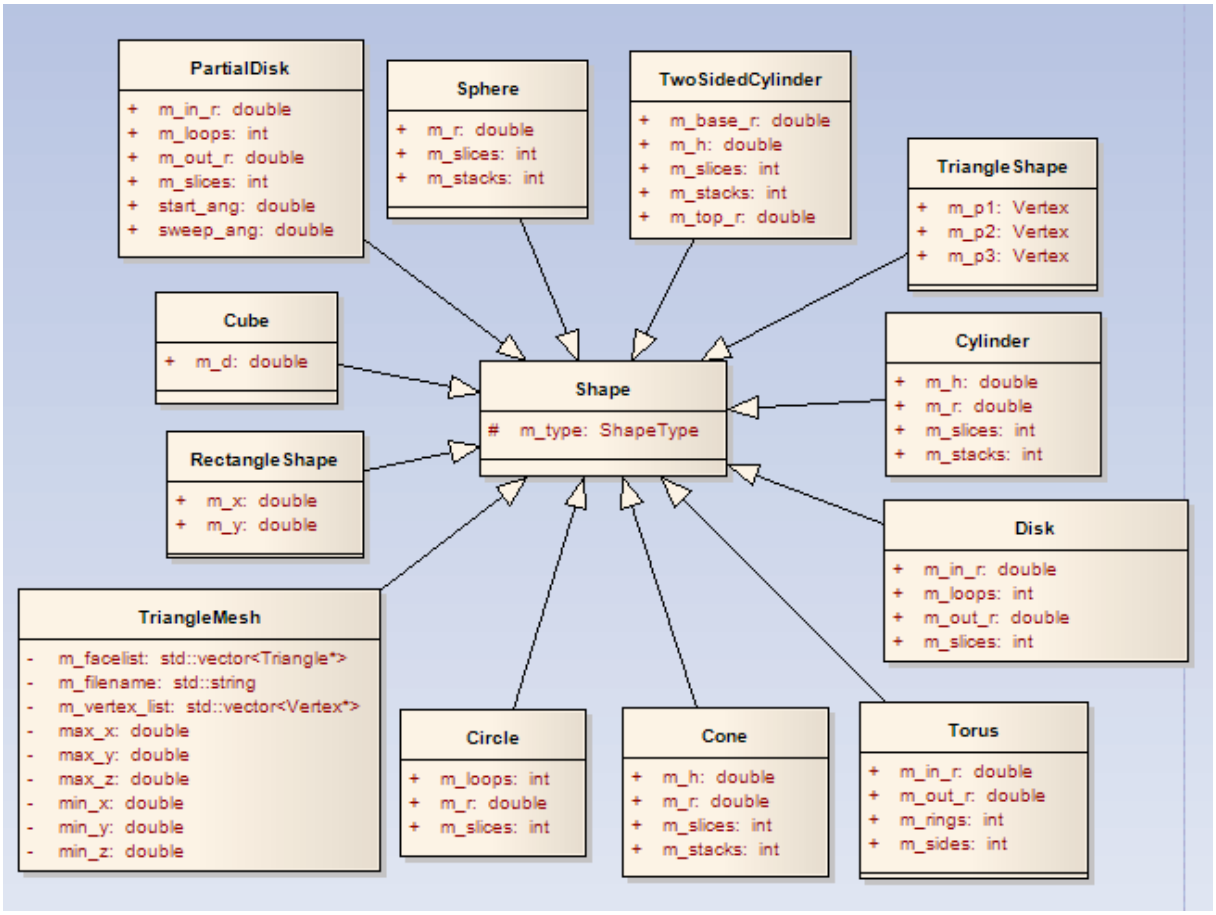


Figure 1. Classes implementing Shape Class

2.1.6 TriangleMesh Class:

That class derived from shape class and it contains a list of triangles, and vertexes. It does not have a geometric shape representation. Any shape can be converted to triangle mesh with some parameters. Triangles in the triangle list references vertexes in the vertex list like indexed face set logic but it is uses references instead indexes. It also has some functions enabling editing of a vertex or a face in its vertex or face list like “getselectedFace()” , “getselectedVertex()”, “clearFaceSelection()” and “clearVertexSelection()”.

2.1.8 Cube Class:

Cube class is representation of geometric cube in scene modeler. It has only dimension parameter. It can be editable with object property window. Cube class can copy itself into a triangle mesh object. This means it can triangulate itself.

2.1.9 Sphere Class:

Sphere class is representation of geometric sphere into scene modeler implementation. Sphere has a radius, number of slices and number of stacks as a parameter. Radius represents the

"dimension of sphere. Slices and stacks are related sensitivity of sphere like slices represents number of vertical lines to draw sphere and stacks represents number of horizontal lines to draw sphere. Sphere class also triangulate itself with copying itself into a triangle mesh object with the sensitivity determined by stacks and slices parameter.

2.1.10 Cylinder Class:

Cylinder class is representation of geometric cylinder into scene modeler implementation. Cylinder has a radius, height, number of slices and number of stacks as a parameter. Radius represent top and bottom radius of cylinder. Height represents the height of cylinder. Slices and stacks are related sensitivity of cylinder like slices represents number of vertical lines to draw cylinder and stacks represents number of horizontal lines to draw cylinder. Loops to draw top and bottom disks assumed to be 1 and it is not parameterized in scene modeler. Cylinder class also triangulate itself with copying itself into a triangle mesh object with the sensitivity determined by stacks and slices parameter.

2.1.11 Cone Class:

Cone class is representation for geometric cone object. It has radius, height, slices and stack parameter. Radius represent the bottom radius, height represent the height of cone. Slices and stacks are related with its drawing sensitivity. Slices represents number of vertical lines, stacks represents number of horizontal lines.

2.1.12 TwoFacetCylinder Class:

Two facet cylinder means cylinder with different radius on bottom and top. It's basically a cylinder with enabling to change bottom and top radius independently. If we change its top radius to zero it acts as a cone.

2.1.13 Torus Class:

Torus is a more complex geometric shape. It has inner radius, outer radius, slides and loops parameters. Inner radius represents the thickness of torus, outer radius of torus represents dimension of torus.

2.1.14 TriangleShape Class:

It is a shape representation of a single triangle. It has two vertexes as parameter. It is enables editing a single face as an object.

2.1.15 Rectangle Class:

Rectangle class is a representation for 2D geometric rectangle. Its parameters are width and height.

2.1.16 Circle Class:

Circle is a representation of 2D geometric circle. It has radius, loops and slices as parameter. Radius represents the radius of circle. Slices represent the number of lines from center to circle at radius. Loops represent the number of single circles to draw whole circle.

2.1.17 Disk Class:

In addition to Circle Class it has editable inner radius and outer radius parameters. If we set inner radius to 0, it acts as a circle.

2.1.18 PartialDisk Class:

In addition to disk class, it has extra editable start and sweep to determine starting angle and finish angle of disk.

2.1.19 Rotation4 Class:

It is implementation of the quaternion concept. Quaternion means rotation along a vector like 45 degree rotation between (1, 3, 5) rotations. This class handles manipulation, addition and applying to vector operations of quaternion. It is widely used in the functional class for geometric calculations. For example in the rotation tool with mouse movement two rotations are calculated along cameras up vector and cameras axis x. To combine these two rotations we use this class. It converts itself to matrix for applying a point.

2.1.20 Transform Class:

Transform class keeps rotation as Rotation4 object, scale as a vector and translation as a vector. It combines itself with another transform means, calculating a new transform with applying two different transforms.

2.1.21 Camera Class:

Camera class is a model for 3D camera. It has up vector, position and target point as parameters. It has also name to identify itself.

2.1.22 TRadiance Class:

TRadiance class is representation for an RGB color. It has red, green and blue components in the 0-1 interval. This means red, green and blue components can take a value between 0 and 1.

2.1.23 Material Class:

Material class is representation for material concept. It has diffuse, secular and ambient colors in the TRadiance type. It also has some parameters related with solidness, shines and transparency. Material class has a flag representing if it has a texture or not. Texture is represented with scale and a path to texture image.

2.1.24 Light Class:

Light class represents a light component of the scene description. It has two type one is point light second is area light. Point light contains only position information and area light contains position and dimension parameters.

2.1.23 Primitve Class:

Primitive class is an abstract class for object definition in the scene description. It has material and some virtual functions like “object_bound()”.

2.1.24 GfxObject Class:

It is implementation for a geometric object containing shape, material and transformation. **Figure 2** shows the attributes of this class. It contains a list of GfxObject references meaning its child object. It also has a reference to its parent. It has name and two type of transformation. One is individual transform is applied only this object itself. Second is public transform, it is applied both this object and its all children.

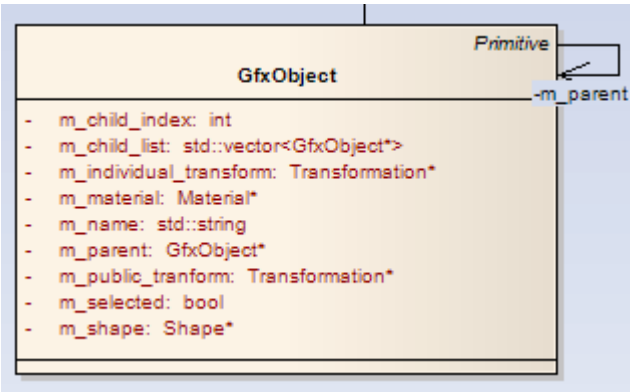


Figure 2. GfxObject Class attributes

2.1.25 Scene Class:

Scene class is representation of scene description with all components. It has a list of GfxObject for geometries, a list of lights and a list of cameras. This class is contains some functions for enabling UI editing. Note that there exist only one scene in scene modeler, all classes having scene objects has its reference.

2.2 FUNCTIONAL CLASSES

There exist two types of functional classes. First types of functional classes are that have UI items like menus, line edit and some other UI components. Classes in this type of functional classes supply functionality with user interactions. SMMenuHandler is a good example for this type of class. It has menu actions and toolbars, it accepts signal from UI items and supply related functionality in terms of signaled action. Second types of functional classes are that do not have UI items. These class commonly used for some specific geometric calculations and manipulations. EditingCamera is an example for functional classes that do not have UI items. It implements Camera mentioned in 2.1.21 and add some extra functionalities to Camera implementations like zoom, in zoom out functions.

2.2.1 EditingCamera Class:

This class implements a storage class named Camera mentioned in 2.1.21. It adds some extra functionality to make enable some camera manipulations with mouse and other user interaction devices. For example, zoom in, zoom out, rotate along target and pan functionalities implemented with related functions in this class. Canvas views, perspective, top, right and front views, uses this camera type to make easy user camera editing.

2.2.2 SMMenuHandler Class:

This class is responsible to handle all UI menu and toolbar actions. In the main window class (in the SceneModeller class) this class is created and references for all UI menu and toolbar actions sent to this class with functions name like “set*Actions()”. Slots to handle UI menu and toolbar actions are also implemented in this class.

2.2.3 ExtrudeDialog Class:

This class implements QDialog class and has a user interface. It has a slider bar. It is used for extruding face. In the face-editing mode if user activate extrude face action and select a face, this dialog appears and with changing slider bar, user can extrude selected face.

2.2.4 CanvasGrid Class:

This class is responsible to draw grid of canvas. Grid can be in XY, YZ and XZ planes. And grid can have 1x or 5x sensitivity. With these parameters this class draws grids with OpenGL commands.

2.2.5 SMCanvasWindow Class:

This class implements QWidget and acts as a container for main drawing widget SMCanvasWidget. It contains SMCanvasWidget and some view related menus like camera manipulation and grid on-off menus. There exist four SMCanvasWindow classes in the scene modeler. Top, perspective, right and front has their individual SMCanvasWindows. SMCanvasWindow also handles mouse actions and take operation in terms of program mode.

2.2.6 SMCanvasWidget Class:

It is a class that draws scene. It implements QGLWidget that is special widget of Qt for OpenGL renderings. It gives a window to openGL rendering. This class acts as a render engine for Scene modeler. It draws all scenes with respect to its individual camera. Each SMCanvasWindow has its own cameras that are type of Editing Camera. It also handles picking for objects, face, vertexes, cameras and lights.

2.2.7 SMIODevice Class:

This is an abstract class for saving scene to a file and loading scene from a file. **Figure 3** shows classes that are implements this abstract class.

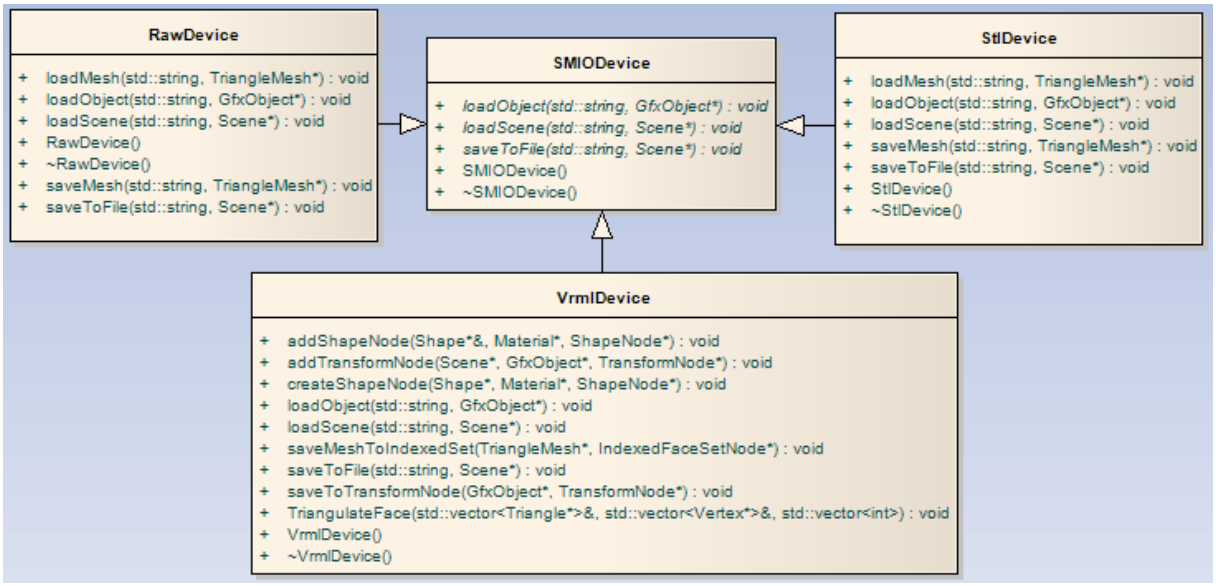


Figure 3. Classes implementing SMIODevice Class

2.2.8 RawDevice Class:

RawDevice class implements SMIODevice and it is responsible to save scene to a raw file with raw format and responsible to read raw formats. Note that raw format contains only triangle information and object names. For this reason this class save only geometric object with triangulating them. While reading, it reads loads scene to triangular mesh object.

2.2.9 StlDevice Class:

This class implements SMIODevice ans it is responsible to save scene to stl file and to load stl files to scene. It also supports only triangulated objects to save and load like RawDevice.

2.2.10 VrmlDevice Class:

This class implements SMIODevice and it is responsible to save scene to vrml file and to load scene from vrml file. It uses open source vrml parser "CyberVRML97" to read and write vrml files.

2.2.11 CameraTool Class:

This class is responsible to handle mouse actions to edit viewing camera. SMCanvasWindow call use this class when camera tools activated. This class makes zoom in, zoom out, rotate and pan camera in terms of the active camera action.

2.2.12 TranslationTool Class:

This class is responsible for translating scene components and geometric items in terms of the program mode. In object mode and in object tree mode, this class translates object, in the camera mode this class translates cameras, in the face mode this class translates faces, in the vertex mode this class translates vertexes and in the light mode this class translates lights.

2.2.13 RotationTool Class:

This class is used to rotate object on the scene. In the object mode or in the object tree mode, it takes a Transformation reference to start rotation and accepts mouse position changes to rotate Transformation.

2.2.14 ScaleTool Class:

This class is used to scale object on the scene. In the object mode or in the object tree mode, it takes a Transformation reference to start scale and accepts mouse position changes to scale Transformation.

2.2.15 CameraPropertiesController Class:

It is responsible to control camera property window. It updates its content after all changes on the current camera. Current camera means EditingCamera of active view.

2.2.16 ObjectTreeController Class:

This class is responsible to control object tree window. It updates its content in terms of object hierarchy after every change on scene. It also handles mouse click event to select objects in the object mode. In the object tree mode it enables and handles mouse drag drop events to change object hierarch relations.

2.2.17 ObjectPropertyController Class:

This class is responsible to handle Object property window. It updates its content every selections. In object mode and in object tree mode it shows selected objects properties and accept user inputs to manipulate object properties. In the light mode it shows the property of selected light and accepts user inputs to manipulate selected light. In the camera mode, it

shows properties of selected camera and accepts user inputs to manipulate camera. In the face mode it shows the property of selected face and it accepts user inputs to manipulate selected face properties. In the vertex editing mode it shows the selected vertex property and it accepts user inputs to change selected vertex property.

3. USER INTERFACE

In this part we firstly describe the window layout of Scene Modeler as descriptions of menus, toolbars, views, and windows. In the second part we mention about some functionalities of scene modeler.

3.1 USER INTERFACE DESIGN

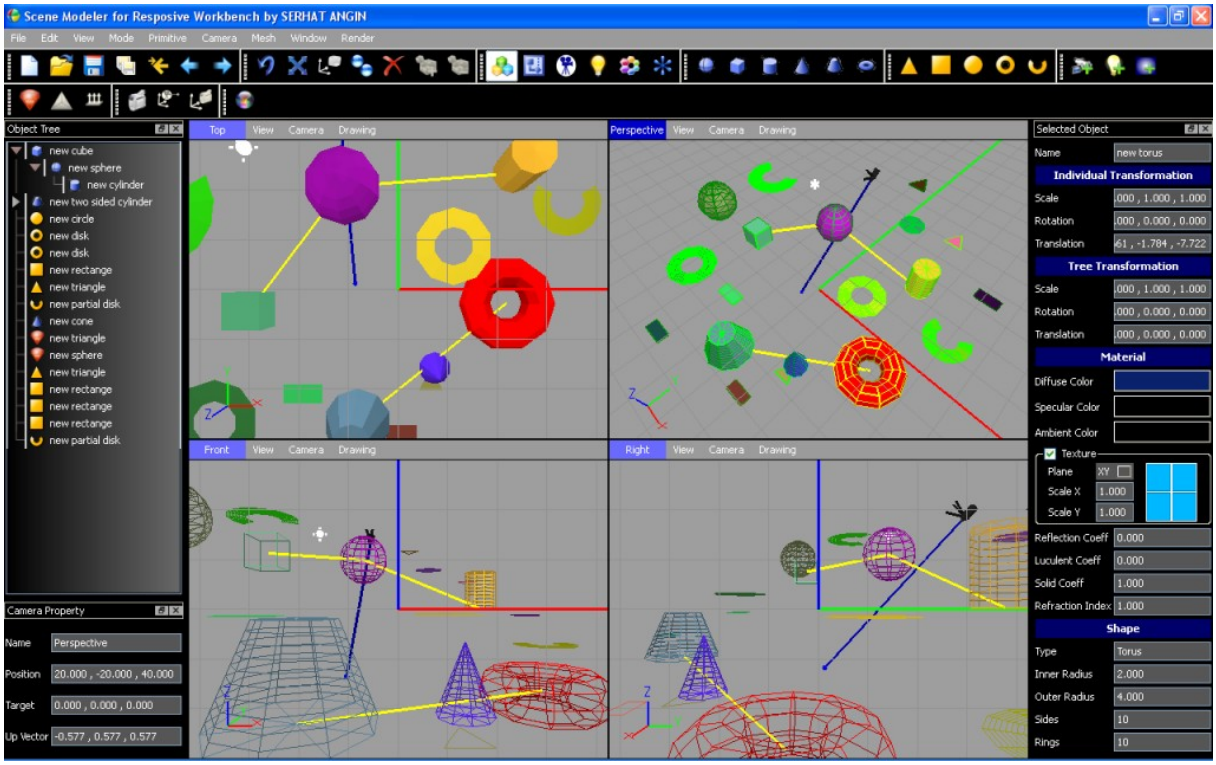


Figure 4. General view for Scene Modeller

In the **Figure 4**, we see the general layout of scene modeler. We can divide main UI items in the general view of scene modeler into menus, toolbars, canvas views and windows. Under menus title there exist file, edit, view, mode, primitive, camera, mesh, and window and render menus. All menus contain specific submenus or actions related with its title. In the second part toolbars, there exist file, edit, mode, 3d objects, 2d objects, primitives, mesh, and camera and render toolbar. These toolbars can be closed and reopened with the help of actions under view menu. In the third classes there exist 4 views of canvas. Each canvas has its default looking direction as top, perspective, front and right. Each canvas supplies a view to scene from different cameras. In the last class windows contains object tree window, camera property window and selected item property window.

3.1.1 Menus

Menus located in the menu bar that is located on the top of window under window title bar. There exist file, edit, view, mode, primitives, cameras, and mesh, window, and render menus on the menu bar. All menus contain its related actions under its title.

3.1.1.1 File Menu:

File menu has actions related with file operations like save, open, import, export and exit to program. In this menu there exist new, open, save, save as, merge, import, export and exit actions. **Figure 5** shows the locations of these actions.

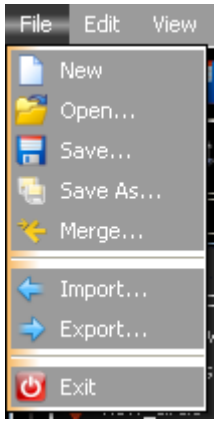


Figure 5. File Menu

3.1.1.1.1 New Action: It creates new scene with basically clearing it.

3.1.1.1.2 Open Action: It opens a file dialog to user to select a file to open. It clears scene before opening a file.

3.1.1.1.3 Save Action: It opens a file dialog to save scene.

3.1.1.1.4 Save As Action: It opens a file dialog to save as scene.

3.1.1.1.5 Merge Action: It opens a file dialog to open file. After user open a file, scene loaded from file merged to scene. This means two objects, lights and cameras into two scenes are combined in a single scene

3.1.1.1.6 Import Action: It opens a dialog to open a file. It merges scene loaded from opened file.

3.1.1.1.7 Export Action: It opens a dialog to save scene.

3.1.1.1.8 Exit Action: It exits from the program. It closes scene modeler.

3.1.1.2 Edit Menu:

Edit menu contains actions that activates some basic editing operations. Some of actions in this menu can be enable or disable in terms of the program editing mode. There exist rotate,

scale, translate, duplicate, delete, add child and remove child actions under this menu. **Figure 6** shows the locations of these actions.

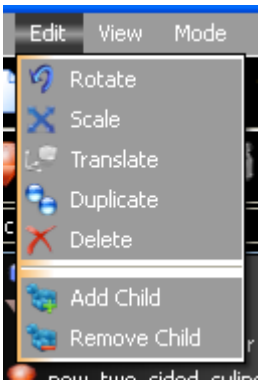


Figure 6. Edit Menu

3.1.1.2.1 Rotate Action: It enables rotation tool for the objects and for the object trees.

3.1.1.2.2 Scale Action: It enables scale tool for the objects and for the object trees.

3.1.1.2.3 Translate Action: It enables translation tool for all items in the scene like light, camera, vertex, face, object and object tree.

3.1.1.2.4 Duplicate Action: It enables duplicate tools for the objects, object trees, lights and cameras. After activating this tool user only click on the item to duplicate it.

3.1.1.2.5 Delete Action: It enables delete tool to objects, object trees, cameras, lights and faces. When delete tool is active, mouse cursor takes a shape as eraser on the canvas views. And user can delete item in terms of program mode by clicking on it.

3.1.1.2.6 Add Child Action: This action is enabled only when the program mode is object tree mode. It enables adding an object to another object as a child. User can click an object that will be a child and click another object that will be parent. After a successful child addition, the new clicked object will be a child again.

3.1.1.2.7 Remove Child Action: It enables remove child actions in the object tree mode. This action is also enabled only in the object tree mode. After enabling remove child tool, user can remove a child with clicking a path between a child and a parent.

3.1.1.3 View Menu:

This menu is designed for closing or opening toolbars that are located on the top of the canvas views. Each actions in this menu has a text with the name of it's related toolbars. And each action under this menu is checkable. This means it an action is checked under this menu, its related toolbars are visible, and vice versa. **Figure 7** shows the layout of these menu actions.

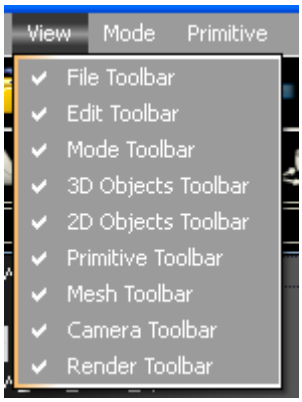


Figure 7. View Menu

File toolbar action close or open file toolbar, Edit toolbar action close or open edit toolbar, Mode toolbar action close or open mode toolbar, 3D objects toolbar action close or open 3D objects toolbar, 2D objects toolbar action close or open 2D objects toolbar, primitive toolbar action close or open primitive toolbar, mesh toolbar action close or open mesh toolbar, camera toolbar action close or open camera toolbar, Render toolbar action close or open render toolbar.

3.1.1.5 Mode Menu:

This menu contains actions that are used to change the editing mode of the program. There exist object, object tree, light, camera, and face and vertex mode actions under this menu. **Figure 8** shows the locations of actions under this menu.

Figure 8. View Menu

3.1.1.5.1 Single Object Action: It changes the editing mode of program to single object mode. With changing the editing mode some actions in menus and toolbar are activated or deactivated.

3.1.1.5.2 Object Tree Action: It changes the editing mode of the program to object tree mode. In this mode all actions like rotations, scale, translations work on the selected object tree.

3.1.1.5.3 Camera Action: It changes the mode of program in to camera editing mode. In this mode camera manipulations are available.

3.1.1.5.4 Light Action: It changes the mode of program in to light editing mode. In this mode light manipulations are available.

3.1.1.5.5 Face Action: It changes the mode of program in to face editing mode. In this mode face manipulations are available.

3.1.1.2.6 Vertex Action: It changes the mode of program in to vertex editing mode. In this mode vertex manipulations are available.

3.1.1.6 Primitive Menu:

Under this menu, there exist actions that create and add new items to scene. Items can be camera, point light, area light, 3D parametric geometric object and 2D parametric geometric objects. **Figure 9** shows the layout of actions under this menu.

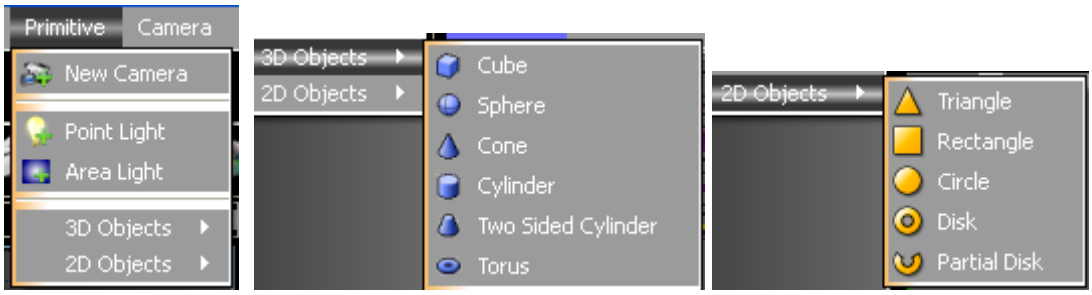


Figure 9. Actions in the primitive menu

3.1.1.6.1 New Camera Action: It adds new camera to scene. If canvases are enabled to draw cameras, this new camera appeared on the canvas views. It adds cameras at the position of origin.

3.1.1.6.2 Point Light Action: It adds a point light to origin.

3.1.1.6.3 Area Light Action: It adds an area light to origin.

3.1.1.6.4 3D objects submenu: This submenu contains actions to add new 3D geometric object at the origin of the scene. There exist cube, sphere, cone, cylinder, two sided cylinder, and torus objects for adding to scene.

3.1.1.6.5 2D objects submenu: This submenu contains actions that add parametric 2D geometric objects to scene at the origin position. User can add triangle, rectangle, circle, disk, and partial disk to scene as 2D parametric geometric objects.

3.1.1.7 Camera Menu:

Under this menu, there are three actions that used to manipulate cameras. Edit cameras, move target and move camera is actions under this menu. **Figure 10** shows the position of actions in the layout of this menu.

Figure 10. Actions in the camera menu

3.1.1.7.1 Edit Camera Action: It opens a camera editing dialog, that enable user to see a list of all cameras defined in the scene and to manipulate these cameras.

3.1.1.7.2 Move Target Action: It enables the tool of move target of camera. In this tool user can select a target point of a camera on the scene with the help of mouse, and translate it. Camera positions does not change and camera rote itself in terms of the new position of target point.

3.1.1.7.3 Move Camera Action: It enables the tool of move body of camera without changing its target point. With this tool user can change the position of camera, and camera rotates itself with unchanged in the target position.

3.1.1.8 Mesh Menu:

Under this menu, there are actions that are related with creation of mesh object or editing the triangle in the mesh object. Convert to mesh, split face, and extrude face are located under this menu. **Figure 11** shows the layout of this menu.

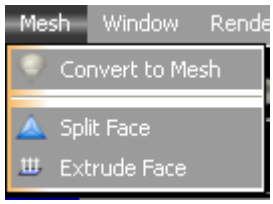


Figure 11. Actions in the mesh menu

3.1.1.8.1 Convert To Mesh Action: It enables convert to mesh tool for objects in the object or object tree mode. After enabling this tool user can click object to convert it to triangulated mesh object.

3.1.1.8.2 Split Face Action: It enables the tool of splitting face. After enabling this tool, user can click under the face to split it into three triangles. This action is active for only face editing mode.

3.1.1.8.3 Extrude Face Action: It enables the tool of extrude face. After enabling this tool, user can click face to extrude it. A extrude face dialog appears after clicking and user extrude face with changing the value of slider on the extrude face dialog.

3.1.1.9 Window Menu:

This menu is designed for closing or opening dockable windows in the scene modeler. There are three dockable windows in our program as object tree window, camera property window

and object property window. With the help of checkable actions under this menu user can close or open these dockable windows. **Figure 12** shows the view of this menu.

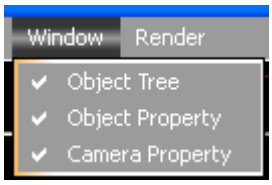


Figure 12. Window Menu

Object tree action close or open the object tree window, object property action open or close the object property window, camera property action open or close the camera property window.

3.1.1.10 Render Menu:

There is only one action under this menu. This is render action. It is for rendering current scene with the camera of current canvas view.

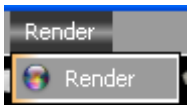


Figure 12. Render Menu

It opens a dialog for setting other parameters of rendering like sky light, anti-aliasing and the window size of rendering.

3.1.2 Tool Bars

Toolbars are located between menu bar and canvas view widgets. Actions are grouped in toolbars as grouping in the menus. All actions in the toolbars are also in the menu. For this reason, we will not deeply explain actions again. We can select and match actions in menus and actions in toolbar by looking their icons. In addition when we keep mouse on the top of action in the toolbar, we see the real text of this action as a tool tip. Note that this text is same as the text of same action in menu bars.

3.1.2.1 File Toolbar:

This toolbar contains actions exactly same as the file menu. **Figure 13** shows the file toolbar.

Figure 13. File Toolbar

3.1.2.2 Edit Toolbar:

This toolbar contains actions exactly same as the edit menu. **Figure 14** shows the edit toolbar.

Figure 14. Edit Toolbar

3.1.2.3 Mode Toolbar:

This toolbar contains actions exactly same as the mode menu. **Figure 15** shows the mode toolbar. Note that the mode of program can be easily understood with looking this toolbar since the action of current mode is heightened.

Figure 15. Mode Toolbar

3.1.2.4 3D objects Toolbar:

This toolbar contains actions that are located in the 3d objects submenu of primitive menus. **Figure 16** shows the 3D objects toolbar.

Figure 16. 3d objects toolbar

3.1.2.5 2D objects Toolbar:

This toolbar contains actions that are located in the 2d objects submenu of primitive menus. **Figure 17** shows the 2D objects toolbar.

Figure 17. 2D objects Toolbar

3.1.2.6 Primitive Toolbar:

This toolbar contains actions exactly same as the primitive menu expect that 2d and 3d submenus. **Figure 18** shows the primitive toolbar.

Figure 18. Primitive Toolbar

3.1.2.7 Mesh Toolbar:

This toolbar contains actions exactly same as the mesh menu. **Figure 19** shows the mesh toolbar.

Figure 19. Mesh Toolbar

3.1.2.8 Camera Toolbar:

This toolbar contains actions exactly same as the camera menu. **Figure 20** shows the camera toolbar.

Figure 20. Camera Toolbar

3.1.2.9 Render Toolbar:

This toolbar contains actions exactly same as the render menu. **Figure 21** shows the render toolbar.

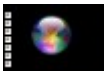


Figure 21. File Toolbar

3.1.3 Canvas Views

Canvas views are main components of scene modeler. There are four canvas views as top view, perspective view, right view and front view. Each canvas has a label showing its name. Canvas views are responsible to show scene from different perspectives and accepts user mouse actions to manipulate scene. **Figure 22** shows the default positions of canvas views.

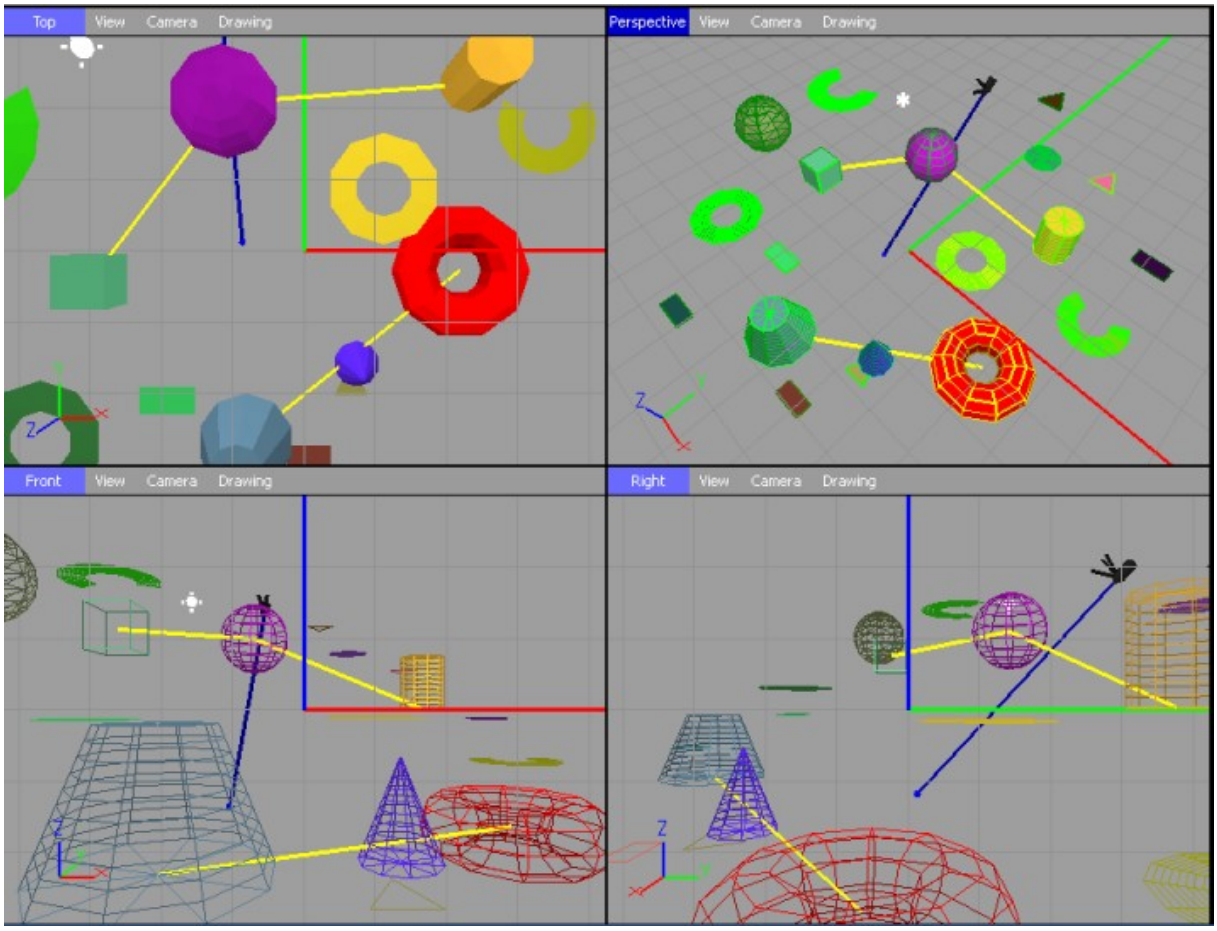


Figure 22. Locations of canvas views.

Each canvas has a title with its name, and the background color of this label is changed in terms of it is active or non active view. In **Figure 22** notice that perspective view is an active view since the background color of its title label is different from others. User can enlarge a view with double clicking a view and can minimize it with also double clicking. Enlarging and minimization can be performed with the help of full screen action under view menu of canvas views. Active canvas view changed with mouse clicking on another canvas view or with mouse wheel event on the other canvas views. Mouse wheel event is used to zoom in and zoom out the camera in the active window. All canvas views are identical and we show details of only one of them.

3.1.3.1 Layout of a canvas view:

A canvas view contains three main parts as title label on the top-left on it, menu bar on the top of it and drawing region under first two parts. Title bar shows the name of canvas view and it is responsible to show which canvas is active or inactive. Active and in active canvas terms are important for mouse event handling and in the rendering operation. When rendering is performed, EditignCamere of currently active canvas view sent to render engine to render scene. Note that there is exactly only one active canvas view at a time. This means if we activate a view by clicking on it, other canvas views are automatically inactivated. Menu bar of a canvas contains actions only related the individual view. We can change the properties of individual canvas view by using the menu bar of this individual canvas view. There exist thee menu under menu bar as view menu, camera menu and drawing menu.

3.1.3.1.1 View Menu of Canvas View: This menu contains actions that are related the viewing property of this canvas view like full screen, grid options and show hide options. **Figure 23** shows the action under this menu.

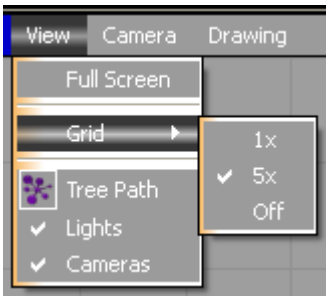


Figure 23. Actions under canvas view menu

Firs action under this menu is full screen action. With this action user can enlarge and minimize canvas like double clicking on it. Second actions are about the options of grid view. It can change the sensitivity of grid and it can on-off this grid. Third action is used to show or hide the path of object trees. This means connections between tree parent and thee nodes. Next action is used to show or hide the lights on this canvas view. The last action under this menu is used to show or hides cameras on the scene.

3.1.3.1.2 Camera Menu of Canvas View: This menu is about the options about the EditingCamera of individual canvas view. It has actions about, resetting, zooming, rotation and panning the viewing camera. Shifting camera is also available under this action. **Figure 24** shows the actions under this menu.

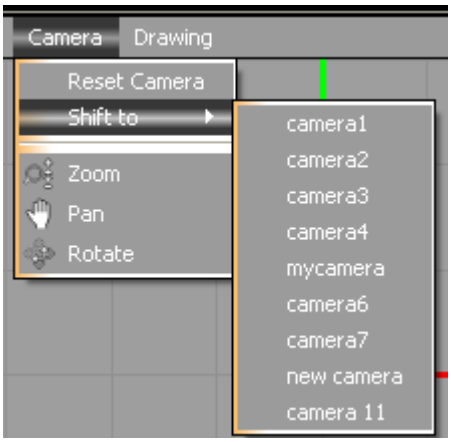


Figure 24. Actions under canvas camera menu

First action under this menu is reset camera action. User can reset the view of scene with the help of this action. Second menu is about shifting camera to a user defined camera. Right side of this action, the list of cameras created by user listed and user can select one of them to shift viewing. Third action is enables zooming tool for this view. After enabling this tool user can zoom in or out by mouse movement on the top of this view. Fourth action is enables the pan tool for this canvas view. User can pan the view by mouse movements on the top of this canvas view after enabling pan tool. The last action is for activating rotation tool for this view. User can rotate camera through target point with using this tool.

3.1.3.1.3 Drawing Menu of Canvas View: This menu contains actions about object drawing options of this view. Use can change the drawing style to wired, shaded or both of them with this menu. Figure 25 shows action under this menu.

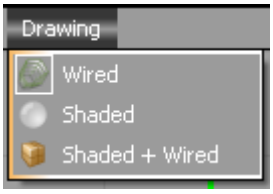


Figure 25. Actions under canvas drawing menu

3.1.3.1.4 Drawing Region in Canvas View: This region is responsible to draw scene with given parameters determined with canvas menus. There exists a translation and scale free axis representation on the bottom-left of this region. These axis representations show the exact direction of x, y and z axis. **Figure 26** shows these axis representations.

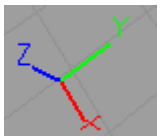


Figure 26. Axis representation on the drawing region

3.1.4 Dockable Windows

Dockable window means, a part of main window that is movable to some other position on the same window, that is resizable , that is closable and that is changeable to stand alone window with itself. There exist three dockable windows on the main window of scene modeler. First one is object tree window on the left side of canvas view as default position. Second one is camera property window on the left bottom side of main window as default position. Third one is the most important one for editing, object property window on the right side of canvas view as default position. Each dockable window has two buttons for closing and changing its mode to stand alone window on the top of it. User can use this buttons for closing this window. With the help of window menu on the menu bar of scene modeler, user can reopen closed window. User can change the position of windows, by dragging and dropping operations with mouse.

3.1.4.1 Object Tree Window:

This is located on the left side of canvas views. It shows the object list defined in the scene in the hierarchical manner. Child parent relations are represented as tree view in this window. **Figure 27** shows the object tree window.

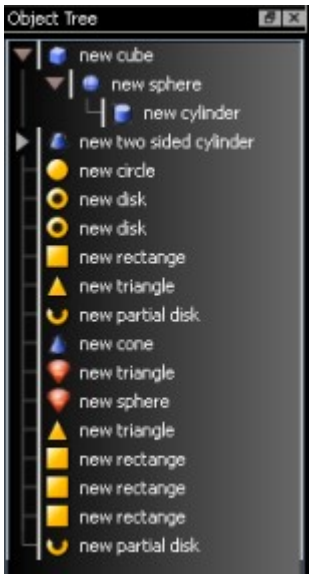


Figure 27. Object tree window

In the object mode object selections can be performed with clicking the items on this tree view. Object tree selections can be performed by clicking on the parent item in tree view when the mode of program is object tree mode. In the object tree mode, changing object hierarchy is also applicable with using this view. User can drag an object that will be a child, and drop it onto an item that will be a parent. If user drops an item in empty region in the tree view, child relation of object related with this item break off.

3.1.4.2 Camera Property Window:

This window is responsible to show properties of editing camera of currently active canvas view. It is read only means user cannot edit editing camera of active canvas with the help of this window. Parameters of camera mean up vector, position and target point and name. If user want to edit the parameter of current viewing camera, camera menu of active canvas view will be helpful. Notice that the names of default cameras are not changeable; these names remain as top, perspective, front and right. **Figure 28** shows the layout of camera property window.

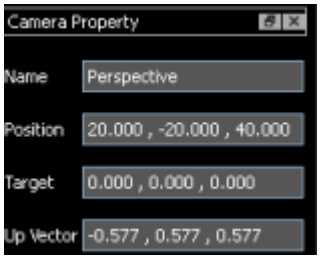


Figure 28. Camera property window

3.1.4.3 Object Property Window:

This window enables more user-friendly functionalities. It shows the property of selected item in terms of the mode of program. In the single object and object tree mode, it shows properties of selected objects including some geometric parameters, and accepts user editing to manipulate selected object. **Figure 29** shows object property window in the single object mode.

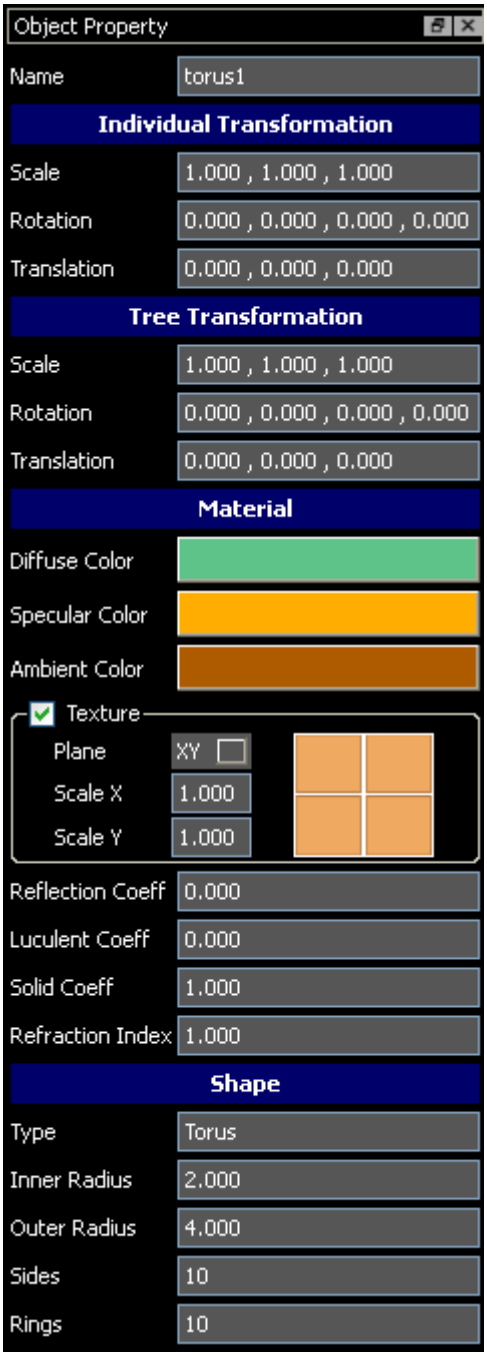


Figure 29. Object property window in single object mode

Text or numbers scene in the line edits can be edited by with the help of keyboard. The color of materials can be edited by clicking on the top of color button, after clicking on the top of color button, a color dialog opened to select new color. To change texture user can use texture

button containing a small icon for current texture, after clicking on the texture button open file dialog opened to select new texture image.

In the camera mode this window shows property of selected camera and accepts user input to change camera parameter. Layout of this window in the camera mode is the same as the camera property window. In the light mode, this window shows the property of selected light and accepts user editing to change properties of selected light. In the face mode this window shows the property of selected face and accepts user inputs to change parameters of selected face. In the vertex editing mode, this windows shows the property of selected vertex and accepts user inputs to change property of selected vertex.

3.1.5 Dialogs

This part includes dialogs that are designed for enabling specific actions like rendering, editing all cameras. These dialogs are shown as a popup and disable scene modeler main window. There are three dialogs in this par as edit cameras dialog, set render parameter dialog and extrude face dialog. Some other dialogs like file open or select color are standard windows dialogs and we will not explain their properties.

3.1.5.1 Edit Camera Dialog:

This dialog is used for camera operations. User sees a list of cameras defined in the scene and edit or delete cameras on the list by clicking on it. User also adds new cameras to scene with this dialog. **Figure 30** shows the layout of this dialog.

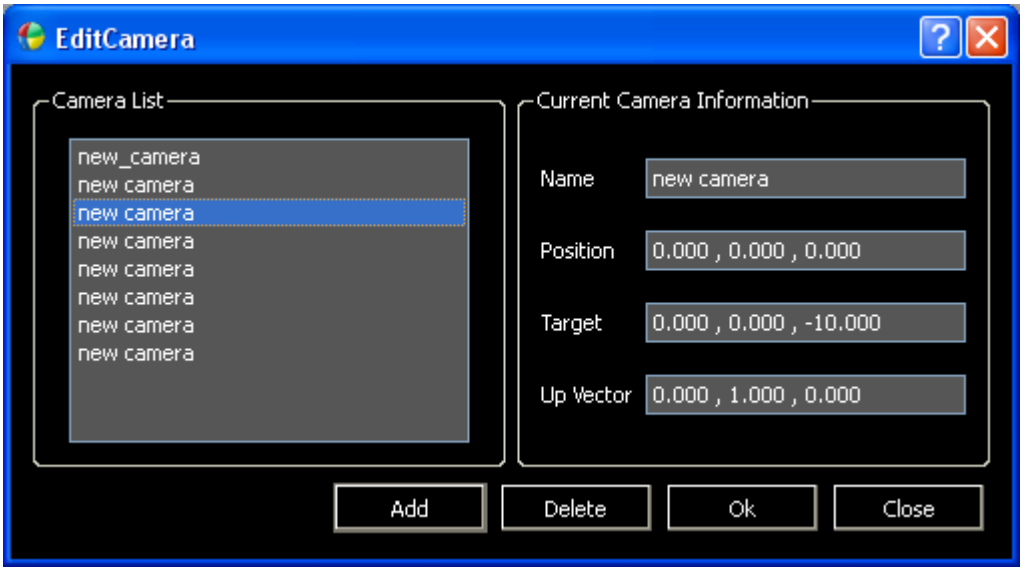


Figure 30. Camera editing dialog

User can select camera with clicking an item on the camera list located left side of dialog, and current camera information part updated with new selected camera. In the camera information part, user can edit the name, position, target and up vector parameters of camera. Notice that when user change one of parameters in position, target or up vector, other parameters are recalculated with camera geometry model and are updated automatically. User can delete selected camera by clicking delete button and add new camera with clicking add button.

3.1.5.2 Extrude Face Dialog:

This dialog is used to extrude face. It has a slider bar to determine the amount of extruding of selected face. **Figure 31** shows this dialog.

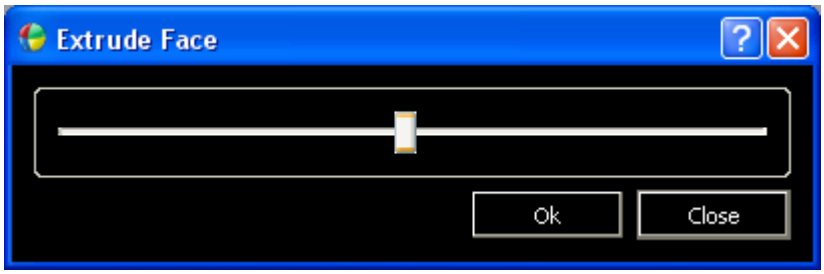


Figure 31. Extrude face dialog

3.1.5.3 Render Options Dialog:

This dialog used to enter parameters for rendering while rendering is performed. User can enable anti-aliasing and sky light with the help of this dialog. User can also enter width and height parameters for rendering with using this dialog. **Figure 32** shows the layout of this dialog.

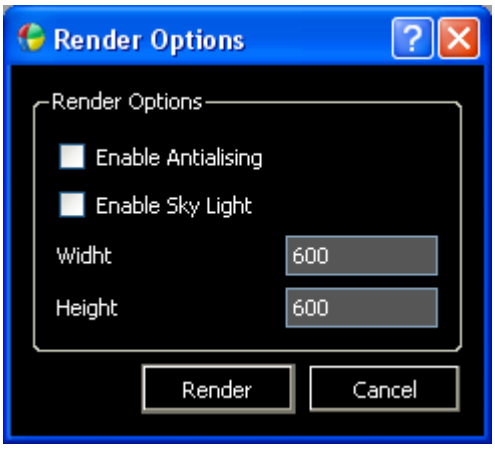


Figure 32. Render Options dialog

User can start rendering with clicking render button or cancel rendering with clicking cancel button.

4. FORMATS

Scene modeler enables to store or reload scene in three formats. First one is VRML 2.0 format. In the VRML 2.0 format we can store all properties of scene as the node structure of VRML. Second file format is raw format. Raw format is designed to store triangulated objects with its vertexes. For this reason we can only store the geometric object and their names with triangulating them. Third format is stl format. In the stl format a triangulated objects can be stored with triangles and their normal. Therefore we can only store objects with in triangle meshes in this format.

4.1 VRML 2.0 FORMAT

VRML is a complex format that has node structure and hierarch. Node means a storage structure including its keywords to store specific type of data. For example there exists a sphere node in VRML that is specific to store only sphere shape in the VRML file. Our data structure is represented in the VRML file format within the combination of nodes. We try to

store our scene as minimum loss in VRML file and we determine the node combinations for our scene descriptions in this manner. Some objects in our scene descriptions cannot be represented exactly in the VRML node structure; in this case we store our objects with some tricky ways including some loss. For example while storing a torus shape, there is no node in VRML structure to keep parameters of torus and we try to store torus as a triangulated version.

4.1.1 Material Representation in VRML:

In data structure of scene modeler, we store diffuse color, ambient color, secular color, and come other parameters for a material. In VRML node structure a material node imported in the shape Node as inside Appearance Node. So if the material does not contain a texture, we can store our material in VRML structure as:

```
Appearance {
        Material {
        ambientIntensity  0.2
        diffuseColor      0.8 0.8 0.8
        emissiveColor     0 0 0
        shininess         0.2
        specularColor     0 0 0
        transparency      0
      }
}
```

Appearance node in VRML can take also a texture node, so if the material has a texture, we can represent a node as fallows:

```
Appearance {
        Material {
        ambientIntensity  0.2
        diffuseColor      0.8 0.8 0.8
        emissiveColor     0 0 0
        shininess         0.2
        specularColor     0 0 0
        transparency      0
      }
      ImageTexture {
        url     []
        repeatS TRUE
        repeatT TRUE
      }
}
```

4.1.2 Shape Representation in VRML:

Shape data structure in scene modeler can be represented as geometry nodes in VRML structure. We can represent some of our parametric geometric shapes with their related nodes in VRML node structure.

4.1.2.1 Sphere Representation: There exists a geometry node named sphere node in VRML node structure. So we can store sphere with this node as:

```
Sphere {
   radius  1
}
```

4.1.2.2 Cylinder Representation: There exists a geometry node named cylinder node in VRML node structure. So we can store cylinder with this node as:

```
Cylinder {
  bottom  TRUE
  height  2
  radius  1
  side    TRUE
  top     TRUE
}
```

4.1.2.3 Cone Representation: There exists a geometry node named cone node in VRML node structure. So we can store cone with this node as:

```
Cone {
  bottomRadius 1
  height       2
  side         TRUE
  bottom       TRUE
}
```

4.1.2.3 Cube Representation: There exists a geometry node named box node in VRML node structure. So we can store cube with this node with setting all dimensions equal as:

```
Box {
  size  d d d
}
```

4.1.2.3 Other Shapes Representation: There is no exactly node representation for other parametric geometric objects. We can store these shapes after converting a triangle mesh with using indexed face set node. These shapes are triangle shape, rectangle shape, disk shape, circle shape, partial disk shape, two faced cylinder shape and torus shape that are triangulated while storing in the VRML format. Therefore we cannot store parameters for these parametric shapes. The structure of indexed face sate is like:

```
IndexedFaceSet {
  set_colorIndex
  set_coordIndex
  set_normalIndex
  set_texCoordIndex
  color             NULL
  coord             NULL
  normal            NULL
  texCoord          NULL
  ccw               TRUE
  colorIndex        []
  colorPerVertex    TRUE
  convex            TRUE
  coordIndex        []
  creaseAngle       0
  normalIndex       []
  normalPerVertex   TRUE
  solid             TRUE
  texCoordIndex     []
}
```

4.1.3 GfxObject Representation in VRML:

We store two different transformations for a single GfxObjects in our data structure. One is individual transformation and the other is object transformation. This enables us to modify both a whole tree and only a root of it. We store a gfxobject structure in the nested transform nodes in the VRML format. Outer transform node represents the public transformations of shape and inner transform node represents the individual transform of gfxobject. We add inner transform node to outer transform node, shape node representing combination of material and geometry representing nodes. Result node combination is seen below:

```
Transform {  // public transformation
  addChildren
  removeChildren
  center           0 0 0
  children         [
           Transform {  // individual transformation
             addChildren
             removeChildren
             center           0 0 0
             children         [
                      Shape {
                           appearance NULL // material representation
                           geometry   NULL // shape representation
                       }
             ]
             rotation         0 0 1  0
             scale            1 1 1
             scaleOrientation 0 0 1  0
             translation      0 0 0
             bboxCenter       0 0 0
             bboxSize         -1 -1 -1
           }
  ]
  rotation         0 0 1  0
  scale            1 1 1
  scaleOrientation 0 0 1  0
  translation      0 0 0
  bboxCenter       0 0 0
  bboxSize         -1 -1 -1
}
```

4.1.4 Camera Representation in VRML:

There is a node named viewpoint node in the VRML representation. These nodes have position and orientation field. We can store our camera in this node representation converting its up vector, direction combinations to Rotation4 representation for orientation. Note that the distance between target and position cannot be stored in this node structure. The structure of this node is following:

```
Viewpoint {
  set_bind
  fieldOfView    0.785398
  jump           TRUE
  orientation    0 0 1  0
  position       0 0 10
```

```
    description    ""
    bindTime
    isBound
  }
```

4.1.5 Light Representation in VRML:

There is a node named point light node in the node structure of VRML. It contains a position and radius for light source. We can directly store a point light with using this node. Problem is storing an area light. We use a tricky way in storing an area light. Here is a trick: if the radius for light sore is 0 this means it is area light, otherwise it represents a point light. The structure of point light node as fallows:

```
PointLight {
  ambientIntensity  0
  attenuation       1 0 0
  color             1 1 1
  intensity         1
  location          0 0 0
  on                TRUE
  radius            100
}
```

4.1.6 Parent-Child Hierarch in VRML:

We use parent-child relations in the data structure of scene modeler between gfxobjects. We can represent a single Gfxobject as the two nested transform nodes in the VRML. While representing parent-child relation in the VRML we add the Group node to outer transform node of gfxobject representation. And then we add representation of child gfxobject to this newly added group node. Node structure will bee like followings:

```
Transform {  // public transformation
  addChildren
  removeChildren
  center           0 0 0
  children         [
           Transform {  // individual transformation
             addChildren
             removeChildren
             center           0 0 0
             children         [
                      Shape {
                           appearance NULL // material representation
                           geometry   NULL // shape representation
                       }
             ]
             rotation         0 0 1  0
             scale            1 1 1
             scaleOrientation 0 0 1  0
             translation      0 0 0
             bboxCenter       0 0 0
             bboxSize         -1 -1 -1
           }
           Group { // child list
```

```
            addChildren
            removeChildren
            children       [

           [ child_Node ]++ // child gfxobject representation or other
                            // gfxobject tree representation
          ]
            bboxCenter     0 0 0
            SFVec3f bboxSize       -1 -1 -1
           }
  ]
  rotation          0 0 1  0
  scale             1 1 1
  scaleOrientation  0 0 1  0
  translation       0 0 0
  bboxCenter        0 0 0
  bboxSize          -1 -1 -1
}
```

4.2 RAW FORMAT

The structure of a raw file is very basic. It contains a list of triangles and optionally it can contain a name of objects before triangle list. The file structure of a raw file is as fallows.

```
[
     object_name
     [
          v1_x v1_y v1_z v2_x v2_y v2_z v3_x v3_y v3_z
     ] ++
] ++
```

4.3 STL FORMAT

STL format contains a normal of triangle and some specific keywords to organize structure in additional to Raw format. In this format we can only keep the triangles, therefore we convert object to triangle meshes to store in STL format. Here is the STL format:

```
[
     solid object_name
          [
               facet normal n_x n_y n_z
                    outer loop
                         vertex v1_x v1_y v1_z
                         vertex v2_x v2_y v2_z
                         vertex v3_x v3_y v3_z
                    end loop
               end facet
          ] ++
     end object_name
]++
```

Note that there is also a binary type for STL file format; Scene Modeler can support only ASCII type of STL format for reading and writing.

5. CONCLUSION

Scene modeler is a 3D scene creating and editing program for the Responsive Workbench project. In this document there is a brief explanation for internal data structures, user interface specifications and file formats supported by scene modeler. In the first part, we divide data structures into two categories as data structure for data storage and data structure for functionalities. In data structure for data storage we explained the specifications and implementation purpose for some important storage classes. In the data structure for functionalities we explained some functional classes used in implementation phase of scene modeler. Note that, this document does not cover all classes in the source code of scene modeler, it covers only important classes. In the second part user interface are described with some useful figures and functionalities of user interface items are briefly explained. In the last part VRML, STL and raw formats are described that are supported by scene modeler. Representation of scene modeler data structure in the VRML node structure is described briefly.

6. DEVELOPMENT ENVIRONMENT

Compiler : Microsoft Visual Studio 2003

Language : C++

Libraries : Qt 4.4.2

: Glut

: CyberVRML97

7. GLOSSARY

3D	Three Dimensions
2D	Two Dimensions
STL	Stereo Lithography
VRML	Virtual Reality Modeling Language
UI	User Interface
ASCII	American Standard Code for Information Interchange
RGB	Red Green Blue
OpenGL	Open Graphics Library

8. REFERENCES

1) The Virtual Reality Modeling Language Specification
http://graphcomp.com/info/specs/sgi/vrml/spec/

2) STL (file format) http://en.wikipedia.org/wiki/STL_(file_format)